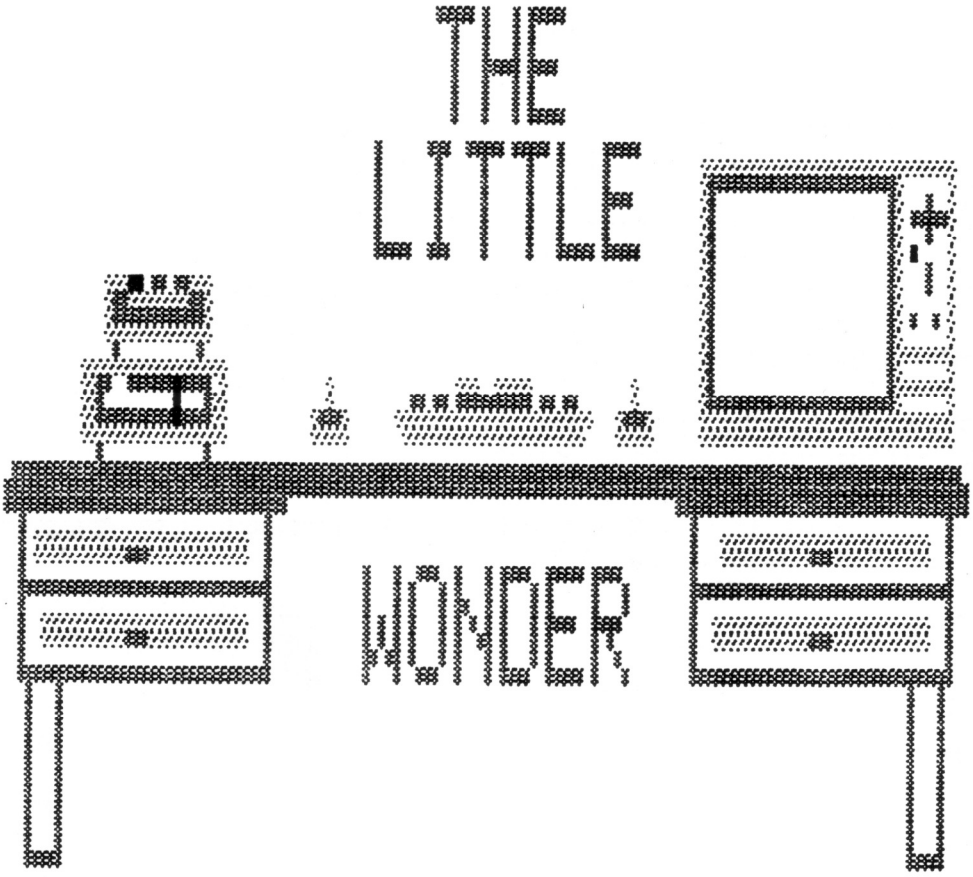
VZ-LINK MAGAZINE

ISSUE No. 29 MARCH 1989

NEW ZEALAND

VZ 300/200

COMPUTER USERS CLUB



NEWS AND VIEWS

Well,it's that time again.This issue is on time for a change.
There was no Feb issue due to my ever increasing work load.
I am sorry this has come about,I have never missed a issue Yet and I hate spoiling a record.

However to makeup for this the April issue should be bigger and have more of everything.
I am still having trouble keeping up with Letters and Software orders.Hopefully these should all be upto date by the Middle of March.

There is still a lot of Membership fees outstanding.
Don't forget these fees are needed to keep this club going.

I am also VERY short of Items for the Newsletter.I would like programs or what have you URGENTLY...

I have will now let you get on with reading this issue.

COMPUTOR BET

COMPUTORBET, written by Marc Atkinson, is a program designed to help in finding the best bet in a horse race. You are first asked to enter in data for the horses. The computer will then analyse your information and tell you what it thinks the best three bets would be. Although it is not guaranteed to pick the winner every time, it will certainly narrow down the feild.

```
5 CLS
10 DIMH$(30):DIMW(30):DIMB(30):DIMT(30):DIMSC(30)
15 DIMP1(30):DIMP2(30):DIMP3(30):DIMP4(30):DIMP5(30)
16 DIMPL(30):DIMPT(30)
20 FORX=1TOH:SC(X)=1000:NEXT
100 PRINTTAB(8)"[illegible]":PRINTTAB(6)"*****************"
101 PRINT:PRINT"THIS PROGRAM WILL GIVE PICKS ON"
102 PRINT"ANY GIVEN RACE. ENTER DATA AS "
103 PRINT"INSTRUCTED. THE COMPUTOR WILL "
104 PRINT"GIVE YOU ITS TOP 3 PICKS . "
105 PRINT:PRINT"--------------------------------"
107 PRINT
110 INPUT"[illegible]";TN$
112 PRINT
115 INPUT"[illegible]";RN
116 PRINT:INPUT"[illegible]";H
118 PRINT:PRINT"[illegible]" :PRINT
119 PRINTTAB(8)"[illegible]":PRINTTAB(8)"[illegible]"
120 PRINTTAB(8)"[illegible]"
121 INPUT"            ";TC
125 GOSUB3000
500 CLS:PRINTTAB(2)TN$;TAB(16)"RACE ";RN
501 PRINT"--------------------------------"
502 PRINT:PRINT"FIELD ENTRY"
510 FORX=1TOH:PRINT
515 PRINT"HORSE NO ";X;:INPUT"  ";H$(X)
520 NEXT
530 GOSUB8000
1000 CLS:PRINTTN$;TAB(14)"RACE ";RN
1005 PRINT"--------------------------------"
```

```
3045 W(C+1)=W(C):H$(C+1)=H$(C)
3050 W(C)=X:H$(C)=Y$
3055 J=J+1
3060 NEXTC
3065 IFJ>0THEN3010
3900 FORX=1TOH
3905 SC(X)=1000-3*W(X)-X
```

```
1010 FORX=1TOH
1020 PRINTX;" ";H$(X):PRINT
1030 INPUT"ENTER WEIGHT IN KG";W(X):PRINT
1060 PRINT"--------------------------------":NEXT
1070 GOSUB8000
3000 CLS:PRINTTAB(8)"[illegible]":PRINT
3002 PRINT
3005 PRINT"HORSE ";TAB(10)"WEIGHT";TAB(21)"SCORE"
3006 PRINT
3010 J=0
3015 FORC=1TOH-1
3020 IFW(C)<=W(C+1)THEN3060
3040 X=W(C+1):Y$=H$(C+1)
4000 PRINTH$(X);TAB(11)W(X);TAB(20)SC(X)
4005 NEXT
4010 GOSUB8000
5000 CLS:PRINTTN$;TAB(14)"RACE ";RN
5005 PRINT"--------------------------------"
5010 FORX=1TOH
5020 PRINTX;" ";H$(X):PRINT
5031 INPUT"ENTER BARRIER DRAW(1-30)";B(X):PRINT
5060 PRINT"--------------------------------":NEXT
5070 GOSUB8000
6000 CLS:PRINTTAB(8)"[illegible]":PRINT
6002 PRINT
6005 PRINT"HORSE ";TAB(10)"DRAW";TAB(21)"SCORE"
6006 PRINT
6010 J=0
6015 FORC=1TOH-1
6020 IFB(C)<B(C+1)THEN6060
6040 X=B(C+1):Y$=H$(C+1):Z=SC(C+1)
6045 B(C+1)=B(C):H$(C+1)=H$(C):SC(C+1)=SC(C)
6050 B(C)=X:H$(C)=Y$:SC(C)=Z
6055 J=J+1
6060 NEXTC
6065 IFJ>0THEN6010
6900 FORX=1TOH
6905 SC(X)=SC(X)-3*B(X)
6907 PRINTH$(X);TAB(11)B(X);TAB(20)SC(X)
6908 NEXT
6909 GOSUB8000
7000 CLS:PRINTTN$;TAB(14)"RACE ";RN
7005 PRINT"--------------------------------"
7010 FORX=1TOH
7020 PRINTX;" ";H$(X):PRINT
7050 INPUT"ENTER LAST 5 PLACES ";P1(X),P2(X),P3(X),P4(X),P5(X)
7060 PRINT"--------------------------------"
7070 NEXT
7080 GOSUB8000
7100 FORX=1TOH
7200 PL(X)=P1(X)+P2(X)+1.5*P3(X)+2*P4(X)+2*P5(X):PL(X)=PL(X)/5
7300 NEXT
7330 IFPL(C)=PL(C+1)THENPL(C)=PL(C)+1
7510 CLS:J=0
7515 FORC=1TOH-1
7520 IFPL(C)<=PL(C+1)THEN7560
7540 X=PL(C+1):Y$=H$(C+1):Z=SC(C+1)
7545 PL(C+1)=PL(C):H$(C+1)=H$(C):SC(C+1)=SC(C)
7550 PL(C)=X:H$(C)=Y$:SC(C)=Z
7555 J=J+1
7560 NEXTC
7565 IFJ>0THEN7510
7600 CLS:PRINTTAB(7)"[illegible]"
7602 PRINT
7603 PRINT"HORSE";TAB(10)"PLACE";TAB(19)"SCORE"
```

```
7607 PRINT
7700 FORX=1TOH
7702 PT(X)=INT(PL(X))
7705 SC(X)=SC(X)-7*PT(X)-7*X
7707 PRINTH$(X);TAB(11)PT(X);TAB(20)SC(X)
7708 NEXT
7800 GOSUB8000
7810 GOTO9000
8000 PRINT:PRINTTAB(8)"[illegible]"
8005 A$=INKEY$
8010 A$=INKEY$:IFA$=""THEN8005
8015 RETURN
9000 GOSUB9400:CLS:PRINTTAB(8)"[illegible]"
9010 PRINT
9015 PRINT"HORSE";TAB(16)"SCORE"
9017 PRINT
9018 FORX=1TOH
9020 IFT(X)=TCTHENSC(X)=SC(X)+30ELSESC(X)=SC(X)-15
9025 PRINTH$(X);TAB(17)SC(X)
9030 NEXT
9040 GOSUB8000
9100 GOTO9600
9200 CLS:PRINTTAB(8)"[illegible]"
9205 PRINT
9210 J=0
9215 FORC=1TOH-1
9220 IFSC(C)<=SC(C+1)THEN9260
9240 X=SC(C+1):Y$=H$(C+1)
9245 SC(C+1)=SC(C):H$(C+1)=H$(C)
9250 SC(C)=X:H$(C)=Y$
9255 J=J+1
9260 NEXTC
9265 IFJ>0THEN9210
9300 PRINTTN$;TAB(14)"RACE ";RN
9305 PRINT"--------------------------------"
9310 FORX=1TOH
9320 PRINT"TOP HORSE ";H$(H)
9325 PRINT"SCORE     ";SC(H)
9330 PRINT
9340 PRINT"2ND HORSE ";H$(H-1)
9345 PRINT"SCORE     ";SC(H-1)
9350 PRINT:PRINT"HIT <E> TO END OR <R> TO RUN"
9355 A$=INKEY$
9356 D$=INKEY$:IFD$=""THEN9356
9357 IFD$="E"THENCLS:END
9358 IFD$="R"THENRUN
9359 GOTO9355
9400 CLS:PRINTTN$;TAB(14)"RACE ";RN
9405 PRINT"--------------------------------"
9410 FORX=1TOH
9420 PRINTX;" ";H$(X):PRINT
9430 PRINT"ENTER FAVOURITE TRACK CONDITION"
9440 PRINTTAB(8)"FAST........1":     PRINTTAB(8)"MEDIUM......2"
9450 PRINTTAB(8)"SLOW........3"
9455 INPUT"     ENTER NO";T(X)
9460 PRINT"--------------------------------":NEXT
9470 GOSUB8000
9500 RETURN
9600 CLS:PRINTTAB(8)"[illegible]"
9610 PRINT:PRINT
9620 INPUT"ENTER PAPER SELECTIONS";A$,B$,C$
9630 PRINT:PRINT"COMPARING CHOICES NOW WITH DATA"
9640 FORX=1TOH
9650 IFA$=H$(X)THENSC(X)=SC(X)+30
9660 IFB$=H$(X)THENSC(X)=SC(X)+25
9670 IFC$=H$(X)THENSC(X)=SC(X)+15
9680 NEXT
9700 GOSUB8000
9780 GOTO9200
```

SERIAL INTERFACES & MODEMS

In this article we intend to look at what a Serial Interface is, What a Modem is, & explain briefly why they are necessary for communication to and from your computer.

Serial Interface

Most of the signals within your computer travel in what is known as parallel mode. In this form each of the 8 bits that constitute a byte travel over a seperate conductor. This is an efficient form of transmission, but one that is not suitable for some external uses, as it requires too many conductors. This requires 9 conductors (8 data conductors & 1 common return path) & is cumbersome for external use, although most printers usa parallel data transfer.

The best method is to use 2 conductors (1 data & 1 return) and send each bit sequentially to the device. This is the function of the Serial Interface. It receives 8 bits as a byte and sends them out one after another. This system works quite well over short distances but, over longer lengths other problems appear. The make up of telephone cable prohibits the use of a digital signal as such on the Public Switch Telephone Network (PSTN). The easiest way to solve this problem is to convert the signal yet again to one that is acceptable to the PSTN. Sound is sent over the PSTN every day, so it seems a logical choice to send our data as sounds (analogue signal). This is the function of a Modem.

Modems

The most common method of converting data is to assign two frequencies to represent the 1's & 0's of our digital signal. Usually a high frequency will represent a 1, while a lower frequency represents a 0. For example, to transmit the letter "A" this is what happens;

1) Serial data arrives as 1000001 (ASCII code for A)
   Note only 7 bits for ASCII code.
2) Modem converts this to high & low tones (H,L)

3) Transmitted as H,L,L,L,L,L,H

At the receiving end the process is reversed and the H's become 1's while the L's become 0's.

This conversion from digital to analogue is called Modulation, while the reverse process is called Demodulation. The word MODEM comes from the contraction of MODulator/DEModulator.

Modem types

All modems fall into two categorise. These are; Acoustic Coupled, and Direct Connect Modems.

Acoustic Coupled

The acoustic coupled modem is a modem with a microphone & speaker to transmit & receive sounds to & from the telephone handset. This is a cheap modem to produce, but its use is limited to areas where background noise is very quiet, as too much noise will mask the tones being sent or received resulting in errors. Also the "speed" of the modem (the rate of transfer of data) is fairly slow, being restricted to around 300 bits per second (300 BPS).

Direct Connection

The direct connect modem as the name implies, is connected via wiring to your telephone line and the tones injected & received direct to the PSTN. The current standard is to use a cord on the modem with a "double Adaptor" type of plug to plug into your phone socket on the wall & then plug your phone into the double adaptor. The direct connect modem allows much faster speeds (up to 2400 BPS in some cases) and is immune to background noises of course.

Modem Facilities

Various models of modems are available, both to buy and to rent from Telecom. Many have extra facilities such as Auto Answer, Auto Dial, multi speed formats etc.

Auto Answer modems have extra circuitry that will detect an incoming call and answer it, automatically supplying a signal to the other modem (known as the originating party) to ready the circuit for transmission.

Auto Dial modems have the capability to automatically dial predetermined numbers, which can be stored in either the modems or the computers memory, or dial numbers entered from the keyboard. To be legallly connected to the PSTN, the modem must not automatically keep trying a number that it does not detect an answer back signal from (this is to avoid annoying telephone subscribers in the event of dialling the wrong number).

Multi-speed modems can operate at several different data transfer rates, including in some cases "split" speeds where the receive & transmit rates are different. An example of this is Videotext where the receive speed at your terminal is 1200 BPS (to update the screen quickly), while your transmit speed is 75 BPS.

Protocols

The format of the data you send and receive has to be the same at both ends of the link otherwise you won't be able to receive or transmit anything. Various forms of "protocol" are in use to ensure correct transmission and for error checking.

In the above description of a modems operation I simplified the system somewhat. The most essential thing is to make sure that the receive & transmit modems are both acting on the same piece of information simultaneously. to do this several extra bits are sent along with the data. These include a "start bit", to ensure that the receiver knows when a new byte is being transmitted, and "stop bits" to let it know that the byte is complete. Also "parity bits are sometimes added for error checking. A parity bit is a bit added such that the number of 1's in the byte is either an even number or an odd nuber, hence the term "even or odd parity".

e.g using the example from the "Modems" section:

"A" is 1000001:	using odd parity & adding a parity bit so that the sum of 1's is an odd number and start & stop bits we transmit;
"A" = 1100000111:	consisting of start bit, ASCII code, parity bit & stop bit.

If the ASCII code in the example had consisted of an odd number of 1's the parity bit would have been a 0, thus still ensuring an odd number of 1's.

This is just one protocol available out of many.

Conclusion

This has been an attempt to give an oversight of the deep, dark world of communications. There are many books on this subject and anyone interested is advised to check out their local library or technical book shop.

STRUCTURED PROGRAMMING
PART 2

by Chris Hobrough.

In Part 1 we saw how a program could be split up into a few major steps and that these steps could be further split up so that we end up with a hierarchy of ever more detailed modules. Now we'll examine the flow of execution within some of the modules and take a look at the sort of formal command structures that are appearing in most modern versions of BASIC. Many of these have come to BASIC from more structured languages such as Pascal and though they aren't available on the VZ, there can be some advantages in knowing how to simulate them.

The flow of execution through a program can take three basic forms, namely sequence, conditional branching and loops. Each of these is fairly simple in itself but it is the way that they are used that often makes the difference between clear simple logic and a mess.

Conditional Branching

Most of the time program statements are executed one after another in a simple sequencial manner. There is a limit to the usefulness of this kind of flow, however, and sooner or later we may need to branch off according to the result of a decision. This will involve an IF statement or an implied one in a more complex command structure.

The VZ offers simple, one line IF .. THEN .. ELSE decisions. The trouble is that sometimes we can't fit all the things we want to do into a single line and so some versions of BASIC offer multi-line IF structures. They look like this:-

```
IF condition THEN
     ................        ' Statement block
ELSE
     ................        ' Statement block
END IF
```

The results of the search option in our video data base program will depend on whether the title requested is in the file. If it's found the search will stop but if it's not, the search will get to the last record, which will contain an End of File marker (EOF). So the index (I) into the array will either point to the required title or to the EOF marker and we can display the results based on a decision about what I is pointing to, using the structure above.

```
11000 ' Display Results ....... ...............................
11002 '
11006                           ' If "EOF" reached
11008 '
11010 IF VT$(I,1) = "EOF" THEN GOTO 11100 ELSE GOTO 11200
  '
11100     PRINT etc             ' Then
  '
11190 GOTO 11300
11200     PRINT etc             ' Else
  '
  '                             ' End If
11300 GOSUB 12000               ' Hold
11310 RETURN
```

You can see that we use a single line IF statement to branch to one of two blocks of code and a GOTO at the end of the first block to jump over the second one. This sort of structure may not seem all that significant but, if carefully followed, it can pay dividends when you get to complex, nested decision PROGRAM

If the decision involves a choice of several options a CASE statement becomes the best way to achieve it.
A CASE statement looks like this:-

```
SELECT CASE expression          CASE item 1
          ............      ' Statement block
     CASE item 2
          ............      ' Statement block
     CASE ELSE
          ............      ' Statement block
END SELECT
```

There can be as many cases as needed and the CASE ELSE is optional. Processing our choice from the list of menu options is an ideal situation for a CASE structure.

```
4000 ' Process Choice ......... ...............................
4002 '
4010                            ' Select case choice (CH$)
4020    IF CH$ = "F" GOTO 4100   '     Find
4030    IF CH$ = "A" GOTO 4200   '     Add
4040    IF CH$ = "D" GOTO 4300   '     Delete
4050    IF CH$ = "L" GOTO 4400   '     List
4060                            ' End select
4100 GOSUB 5000                 '     Find title
4110     GOTO 4500
4200 GOSUB 6000                 '     Add title
4210     GOTO 4500
4300 GOSUB 7000                 '     Delete title
4310     GOTO 4500
4400 GOSUB 8000                 '     List all titles
4500 RETURN
```

If you have Extended BASIC then the ON .. GOTO command can often be used to achieve the same results and sometimes a simple list of IF's is all that's necessary, provided they each fit on one line. Again, a little discipline pays off, especially when things get complicated.

Loops

When program execution branches backwards to a point already passed and then proceeds through the piece of code again you have a loop. This also involves a decision, or should do, and since the same decision will have to be made each time the loop is repeated, there must be something inside the loop which can change the status of the condition that is being tested.

Loop structures can be of two types: pre-test loops and post-test loops. The WHILE ... NEXT structure in our main control module is an example of a pre-test loop because the condition is tested before the body of the loop is executed each time. Rather than look at that again we'll have a look at the "Find Title" routine from the menu options.

We use a loop here so we can keep searching for different titles without going back to the main menu each time. But we also need an escape route, so we test for "ZZZZ" being entered as a signal that the user has finished with this option.

```
5000 ' Find Title ............. ...............................
5002 '
5010 GOSUB 9000                 ' Get user's request
5020 IF TL$ = "ZZZZ" GOTO 5070  ' While request is not "ZZZZ"
5030     GOSUB 10000            '     Title search
5040     GOSUB 11000            '     Display result
5050     GOSUB 9000             '     Get user's request
5060 GOTO 5020                  ' Next
5070 RETURN
```

We've ended up with a series of sub-routines again and, as before, we need to prime the structure by calling the routine which actually gets the user's request, before getting to the WHILE loop. The test is done with the IF statement which has its logic reversed compared to the WHILE statement. WHILE "ZZZZ" hasn't been entered we search for the request and display the result. Then we get a new request and GOTO the test again. IF "ZZZZ" was entered we jump the loop.

The other type of loop structure is the post-test loop which, as its name suggests, has the condition tested after the body of the loop. The most common form is the REPEAT ... UNTIL structure. The "Hold" routine at the end of "Display Result" will illustrate a REPEAT loop. This routine simply puts a prompt at the bottom of the screen and holds the display until the user is ready to continue.

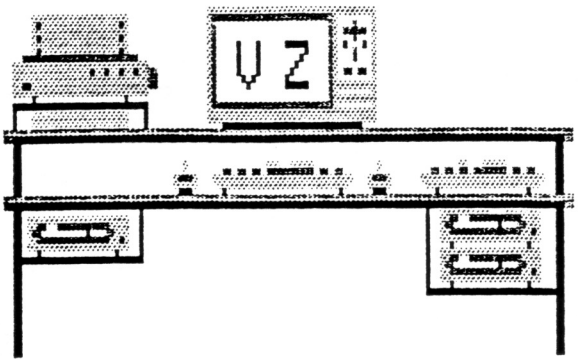
```
12000 ' Hold ................. ............................
12002 '
12006 '                         ' Print user prompt
12008 '
12010 PRINT@ 483,"PRESS <RETURN> TO CONTINUE";
12012 '
12020                           ' Repeat
12030     A$ = INKEY$           '     Get character
12032                           ' Until character is <RETURN>
12038 '
12040 IF A$ <> CHR$(13) GOTO 12020
12050 RETURN
```

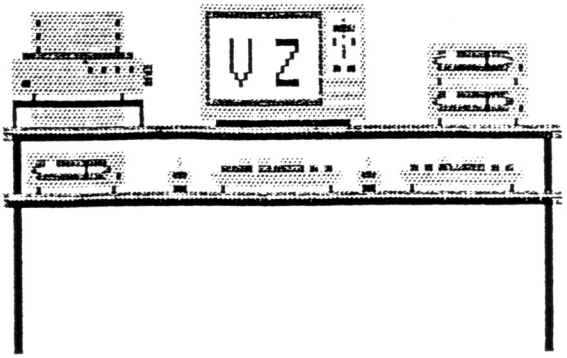
The test is now after the body of the loop, which in this case is a simple INKEY, and the REPEAT line is just a REM for clarity. The main difference between these two forms of loop structure is that a post-test loop must always execute the body of the loop at least once in order to get to the test, whereas a pre-test loop need not go through the loop at all if the test fails the first time. This makes the pre-test form slightly more useful. There is only one loop structure on the VZ and that's the FOR .. TO ..NEXT structure which can be seen as a special case of a WHILE loop.

You may wonder why you need bother simulating structures that the VZ doesn't have. The fact is that you probably already do, perhaps without realizing it, and all that we're really doing is formalizing them and giving them names. Once the concepts are familiar to you they will become automatic. Whenever a certain type of situation arises, you'll recognize the need for one of these structures, rather than keep "re-inventing the wheel". In addition, when you see a WHILE, for instance, in a REMark, you immediately know what the next block of code is about.

In the next part of this article we'll look at the importance of proper documentation and presentation.

DRIVE PROBLEMS BY JOE LEON





DISK DRIVE PROBLEMS CAN TAKE MANY FORMS FROM A BAD BATCH OF DISKS TO ELECTRICAL INTERFERENCE JUST TO NAME A COUPLE. I'M SURE MANY DRIVE USERS HAVE COME ACROSS THE DREADED DISK I/O ERROR AND BEEN AT A LOSS ON WHAT TO DO. LIVING IN A SMALL HOUSE HAS IT'S DRAWBACKS LIKE HAVING TWO COMPUTER SYSTEMS ON THE ONE TABLE.

AS EDITOR I SEND AND RECEIVE QUITE A FEW DISKS AND USED TO HAVE LOTS OF PROBLEMS WITH DISK I/O ERROR ON MY SYSTEM. PERSONS SENDING ME DISKS ASSURED ME NO PROBLEMS AT THEIR END. I PRESUMED DIRTY HEAD AND USED A HEAD CLEANING DISK WITH DISASTROUS RESULTS. (SEE HEAD CLEANING DISK UPDATE ELSEWHERE IN THIS ISSUE).

CLEANING HEAD DID'NT SEEM TO MAKE ANY DIFFERENCE SO HAD TO LOOK FOR ANOTHER CAUSE AND IT WAS SO OBVIOUS THAT I COULD'NT SEE THE FOREST FOR THE TREES, I SIMPLY FORGOT ABOUT IT. THE DRIVES WERE SIMPLY TOO CLOSE TO THE TV, LESS THAN 5 INCHES AND THE MAGNETIC FIELD CREATED BY THE TRANSFORMER IN THE TV WAS AFFECTING PROPER DRIVE OPERATION.

THE SOLUTION WAS OBVIOUS, MOVE THE DRIVES AS FAR AS POSSIBLE FROM THE TV. SOLVING ABOVE PROBLEM CREATED ANOTHER BY REALISING LEADS ATTACHED BETWEEN DRIVE AND DISK CONTROLLER WERE TOO SHORT FOR THE JOB.

ANOTHER PROBLEM WAS LACK OF ROOM ON THE TABLE FOR MY DRIVES DUE TO THE KIDS COMMODORE C64 COMPUTER AND DRIVE SHARING THE TABLE, (THEY'LL HAVE TO GO, C64 TOO).

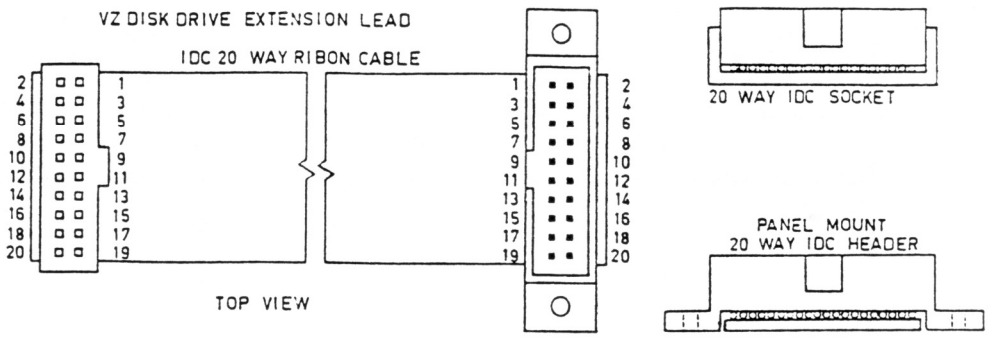
THE DRAWING ON THE RIGHT SHOWS THE DRIVES NEXT TO THE TV AND IT'S ONE PLACE DRIVES SHOULD BE KEPT AWAY FROM WHILE THE DRAWING ON THE LEFT DEPICTS MY PRESENT LAYOUT WHICH HAS CURED THE DISK I/O ERROR.

THE VZ DRIVES NO LONGER SIT BESIDES THE TV, BUT HAVE BEEN MOVED TO A SHELF BELOW THE TABLE AS HAS THE C64 DRIVE. EXTENSION RIBBON CABLE LEADS WERE CONSTRUCTED TO ALLOW DRIVES TO OPERATE ON A SHELF BELOW THE TABLE.

IF YOU HAVE PLENTY ROOM ON YOUR TABLE AND YOU HAVE NO PROBLEMS WITH YOUR DRIVE/S THEN YOU'RE LUCKY AND THE EXTENSION CABLE IS NOT FOR YOU. IF YOU DO GET DISK I/O ERRORS THEN TRY MOVING DRIVE/S AS FAR AWAY AS POSSIBLE FROM THE TV.

ALSO ONE SIDE OF THE TV USUALLY HAS A STRONGER MAGNETIC FIELD THAN THE OTHER DEPENDING IN WHICH SIDE THE TRANSFORMER IS MOUNTED. SOMETIMES IT'S SUFFICIENT TO MOVE DRIVE/S FROM ONE SIDE TO THE OTHER.

DRIVE EXT LEAD BY JOE LEON . .



CONSTRUCTING RIBBON CABLE EXTENSION LEAD :-

PARTS LIST - 1 OFF 20 WAY IDC FLAT RIBBON SOCKET
1 OFF 20 WAY IDC FLAT RIBBON PANEL MOUNT HEADER
1 OFF 20 WAY IDC FLAT RIBBON CABLE - 16" (40CM)

ASSEMBLY IS FAIRLY SIMPLE, JUST FOLLOW THE DIAGRAM. INSERT THE RIBBON CABLE INTO THE SLOT AT BOTTOM OF HEADER/SOCKET MAKING SURE CABLE IS SQUARE AND PUT IT DOWN ON A HARD SURFACE LIKE A BREADBOARD. PLACE A SMALL PIECE OF WOOD ON TOP OF HEADER/SOCKET AND USING A SMALL HAMMER HIT THE WOOD TILL THE CLAMP HAS LOCKED THE CABLE IN PLACE.

REPEAT FOR OTHER END MAKING SURE THE HEADER/SOCKET ARE THE RIGHT WAY ROUND. DO NOT MAKE THE RIBBON CABLE LONGER THAN NECCESSARY TO AVOID INTRODUCING TOO MUCH CAPACITANCE WHICH CAN AFFECT DRIVE OPERATION.

I'VE BEEN USING EXTENSION CABLES WITH MY DRIVES FOR A WHILE NOW AND HAVE EXPERIENCED NO PROBLEMS SO FAR. REMEMBER, CHECK ALL YOUR WORK AS A MISTAKE CAN SPELL DISASTER.

NOTE - WHEN PURCHASING THE RIBBON CABLE MAKE SURE IT'S IDC (INSERTION DISPLACEMENT CONNECTOR) TYPE AS IT HAS THE CORRECT SPACING TO FIT IN THE METAL SLOTS FOR CLAMPING. IF YOU CAN'T GET 20 WAY RIBBON CABLE THEN THE NEXT ONE WIDER WILL DO AS IT'S SIMPLE TO REMOVE THE EXESS STRANDS.

ALSO YOU MAY HAVE TROUBLE GETTING THE PANEL MOUNT IDC HEADER AS IT'S A BIT HARD TO COME BY. DO NOT GET THE HEADER WHICH HAS PINS AT BOTTOM WHICH ARE DESIGNED FOR PRINTED CIRCUIT MOUNTING AND NOT SUITABLE FOR THIS APPLICATION.

THERE ARE TWO MORE WAYS TO ACHIEVE A LONGER LEAD. ONE IS TO REMOVE ENTIRE CABLE FROM DRIVE AND REPLACE WITH LONGER ONE. THE OTHER IS A BIT SIMPLER AND INVOLVES PUTTING AN IDC PANEL MOUNT HEADER AT REAR OF DRIVE AS PER DISK CONTROLLER. THAT WAY THE LEAD IS ENTIRELY REMOVABLE AND MAKES FOR EASIER TRANSPORTATION.

WHICHEVER WAY YOU CHOSE TO CONSTRUCT EXTENSION CABLE I MUST STRESS THE IMPORTANCE OF BEING CAREFULL AS YOU ARE DOING IT AT YOUR OWN RISK. THIS ARTICLE IS MERELY A GUIDE AND IF YOU HAVE ANY DOUBTS ON YOUR ABILITY THEN DON'T ATTEMPT CONSTRUCTION.

BIT - BY - BIT

this month we present some short graphics-handling assembly routines. We'll be looking at some screen scrolling routines in particular, and these should help to form the basis for a BASIC or machine code graphics program.

One very useful function not found in the graphics commands of BASIC is a scroll instruction. While the text screen scrolls automatically, BASIC programs are virtually restricted to using a stationary graphics screen. The slowness of BASIC means that writing a scroll routine in BASIC is impossible, though in Assembly the routine is short and efficient.

When writing code which accesses graphics RAM, we have to consider what is known as 'raster'. As was explained in our article on the video display generator (VDG), the sreen is 'scanned' from left to right to produce the display. No doubt you have noticed the terrible 'flickering' type of interference produced on some graphics games - the TV looks as though it's picking up RF interference from a lawn mower! This is actually caused by impropper handling of the graphics system.

The problem arises when any location within graphics RAM is accessed during a screen scan. Thus to eliminate the eye-irritating effects, all access to graphics RAM must take place between screen scans. In this way the graphics memory is updated before it is displayed on the screen. Luckily (or perhaps by good design) the VDG produces its interrupt signal to the Z80 CPU at the conclusion of each active display area scan, so this can be used to time graphics access correctly.

Now let's consider a practical example. Many graphics packages require that the graphics screen be scrolled. Games such as Zevios, Morgoth and Road Warrior all make use of such scrolling. However, screen scrolling is not feasible with BASIC, but can be implemented easily from machine code by the use of bloack loading.

Considering first vertical upward scrolling, the following routine is all that is required:

```
001 VSCL LD HL,28704
002      LD DE,28672
003      LD BC,2016
004      LDIR
005      RET
```

Line 001 sets the HL reg. to point to the first 'read' address, while the next line sets DE to the first write address LDIR then transfers a block of 2016 bytes (set by BC) upward in memory. Effectively, each byte of graphics RAM is transfered upward (up the screen) by moving each to the address 32 bytes lower in RAM. Thus the screen is scrolled upwards by one line. If you execute this routine, you'll notice that the flicker problem is quite severe. We need to make the program wait for an interrupt before moving the RAM. This is simply acieved by inserting a HALT instruction before the LDIR. When the CPU reads a HALT, it suspends operation until an interrupt is received. Thus the routine waits for an interrupt until it proceeds. Although this slows down the routine, the flicker is greatly reduced

The complete routine now looks like this:

```
001 VSCL LD HL,28704
002      LD DE,28672
003      LD BC,2016
004      HALT
005      LDIR
006      RET
```

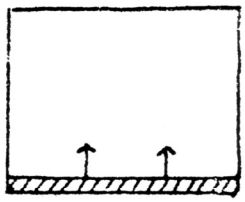
Here's the BASIC version:

```
10 T=PEEK(30897)+256*PEEK(30898)
20 T=T-20:POKE30898,INT(T/256):POKE 30897,T-PEEK(30898)*
   256
30 FOR Z=T+1 TO T+13: READ A: POKE Z-65536,A: NEXT
40 T=T+1:POKE 30863,INT(T/256):POKE30862,T-PEEK(30863)*
   256
50 MODE(1)
.
.
.
100 X=USR(0): REM SCROLL UP SCREEN
```

The program POKEs in the machine code program at the top of memory. Place this routine at the start of your program, and whenever you wish to scroll the screen, simply enter a: X=USR(0). Of course the program is not yet complete. The DATA lines are still to be added, but before we do so, we'll look at some variations on the routine. Below is a couple of alternate DATA lines for either vertical or horizontal scrolling.

Obviously, to scroll the screen to the left, each address must be transfered into the one preceeding it. All that needs to be changed in our routine are the pointers HL & DE. HL must be loaded with 28673, while DE remains 28672. Note that as the screen is arranged with one byte controlling four pixels we are effectively moving the screen four places to the left at a time, though if done quickly enough the display moves smoothly. Note also that with both versions, the bottom, or right column (whichever is the case) will not be affected by the scroll. Your game or whatever must replace this area with the new line to come onto the screen.

DATA LINES FOR ABOVE PROGRAM:

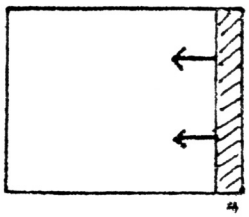


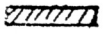
For vertical Scroll Upward:

```
200 DATA 33,32,112,17,0,112,1
210 DATA 224,7,118,237,176,201
```

For Horizontal Scroll 4 places to left:

```
200 DATA 33,1,112,17,0,112,1,255,7
210 DATA 118,237,176,201
```



 – Area to be cleared

SOFTWARE FOR SALE

We now have the following software available for sale.
All prices shown include postage and packing anywhere in New Zealand.
Please make sure that the memory of your VZ is greater than or equal to the memory required by the software that you wish to purchase.
Memory sizes indicated do not include video ram.

ALL PRICES ARE SHOWN IN NEW ZEALAND DOLLARS.

BLACKJACK ROYALE	16K	$15.00	HAUNTED MANSION	22K	$15.00
CASTLE GREYSTONE	32K	$15.00	KNIGHTS QUEST	32K	$18.00
DATA	16K	$18.00	NAMESTAR/SHOPSTAR	22K	$15.00
FBI-2001	31K	$18.00	SCOTLAND YARD	16K	$ 8.00
GALACTIC EMPIRES	16K	$15.00	TRIVIAL CULT	16K	$ 8.00
GRAPHSTAR	22K	$24.00	VZ MONOPOLY V2.0	22K	$18.00

PLEASE MAKE ALL CHEQUES AND MONEY ORDERS PAYABLE TO-

AUCKLAND VZ 300/200 COMPUTER CLUB.

VZ DISASSEMBLER FOR SALE

This Disassembler is written by Peter Hickman.
And is also distributed as the "Laserlink Disassembler".

Peter is offering this very fine program to user group members for NZ$24.00.
All orders can be placed through this club,as peter has offered a cheaper price for bulk purchases.

The program is available on tape or disk.And comes in a 4 program version and
You choose which section you use depending on what memory you have.

ALL THESE PROGRAMS CAN BE DEMONSTRATED AT THE WORKSHOP IF REQUESTED

P.O.BOX 1972 C.P.O AUCKLAND N.Z

(C) AUCKLAND VZ 300/200 COMPUTER USER'S CLUB 1986

ALL MATERIAL IN THIS NEWSLETTER IS SUBJECT TO COPYRIGHT..
CONTRIBUTED PROGRAMS & ARTICLES CAN BE REPRODUCED ON THE UNDERSTANDING THAT THEY ARE FOR THE PRIVATE USE OF SUBSCRIBERS ONLY.
COPYRIGHT IS ALWAY'S RETAINED BY THE AUTHOR.
THIS CLUB WILL NOT BE HELD RESPONSIBLE FOR THE BREECHING OF COPYRIGHT CAUSED BY THE PUBLICATION OF ANY MATERIAL SENT IN UNDER A FALSE NAME.

YOU ARE WELCOME TO PHONE ME AT WORK IF YOU HAVE PROBLEMS